Recovery of Data Model from COBOL Programs
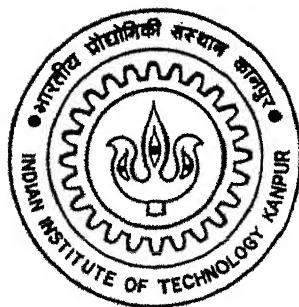
by

S. Suresh Kumar

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1996

Recovery of Data Model from COBOL Programs

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

S. Suresh Kumar

to the

DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
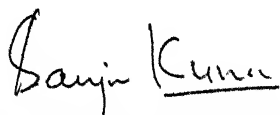## INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
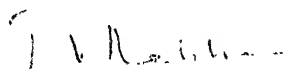
April, 1996.

C SE - 1996 - M - KUM - REC

CERTIFICATE

This is to certify that the work contained in the thesis entitled " Recovery of Data Model from COBOL programs" by "S. Suresh Kumar" (Roll No: 9411132), has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

Dr. Sanjeev Kumar Aggarwal
Associate Professor
Department of Computer
Science & Engineering,
Indian Institute of Technology,
Kanpur.

Dr. T. V. Prabhaker
Associate Professor
Department of Computer
Science & Engineering,
Indian Institute of Technology,
Kanpur.

April, 1996

Acknowledgmens

Abstract

Software re-engineering of a system is a one-to-one transformation of its functions and data into different structures, possibly with different languages and in different environments. The resulting system's functionality should be the same as that of the subject system. One such problem addressed here is the transformation of an existing file based system, implemented in COBOL, to a relational database environment. This transformation is mainly required because the RDBMS systems are better containers of data and provide better accessibility of data than legacy COBOL based systems. This re-engineering process involves two tasks. The first task is, data model re-engineering and the second is restructuring the application programs to C embedded with SQL. The process of re-engineering is too complex to do manually. An automatic migration tool is indispensable for such a task. SQLC is one such tool which has been developed during this thesis. In this thesis, we discuss the conceptual frame work in re-engineering the data model from COBOL programs and subsequent transformation of this model to relational environment.

Contents

.

iii

List of Figures

Chapter 1

Introduction

The information systems of many organizations keep the data in files supported by the operating system. These systems have a number of application programs which work on the data, in response to the needs of the organization. As the need arises, new application programs and new data files are added to the system. These application programs are mostly written in COBOL, because of its rich file handling capabilities and its English like syntax which is highly readable.

The above system has many disadvantages. It has problems like data redundancy and inconsistency, difficulty in accessing data, data isolation, concurrent access anomalies, security problems, integrity problems, etc.. These problems exist mainly because the system does not provide a single integrated view of data, and it does not explicitly specify the relationships among data items. To eliminate the problems in managing data with conventional file systems, to manage complex sets of files and to support data modeling, many database management systems have been developed. These systems support Relational, Network or Hierarchical data models. The Relational model is the most popular among the three models.

The Relational model is very flexible in expressing the relationships among data. In this model, it is possible to change the logical model without affecting the application programs. The database management system that supports the relational model is called relational database management system(RDBMS). The

advantages of RDBMS are that it provides a single abstract data structure, viz., record, relates records by the values they contain, and provides SQL, which is a non-procedural query language for database management. SQL eliminates the need of writing separate application programs for each possible and necessary query.

Nowadays the information systems are being developed in RDBMS environment because of the many advantages it offers. However most of the organizations are continuing with the existing systems because they have already spent a lot on these systems and developing new software is expensive. But maintenance of these systems is consuming an enormous amount of M.I.S.(management Information System) resources. Despite this level of expenditure, the organizations are often not getting the correct information because of weaknesses in data management with conventional file systems. It is also difficult for the external world to access data, due to mismatch in technology. If these systems are migrated to a relational environment the weaknesses due to data management with file systems can be resolved. So it is very important to automatically re-engineer these systems to an RDBMS environment, where it can make avail of better data management facilities till a new system in RDBMS is developed.

1.1 The Problem

The problem is that of building a tool which will automatically map a classical information system, to an RDBMS system. In some cases where automation becomes difficult a user skilled in the classical system must work in tandem with the tool. The new system should not change either the functionality of the existing system or its external behavior. For the problem described, the RDBMS is INGRES supporting C with embedded SQL.

The Re-engineering of these systems involves the following steps:

(1) Extracting the data model from the existing system and transferring it to the relational model.

2

(2) Installing the database with the help of the relational model.

(3) Converting file access statements to SQL queries and other statements to 'C'.

The main emphasis of the current work is to automate the re-engineering process to the maximum extent possible.

1.2   Related Work

COBSQL: It is a tool developed by P.L Srinivas [Sri93] for transformation of COBOL programs to COBOL embedded with SQL programs. This tool will replace the file accessing statements in the COBOL program with equivalent SQL statements.

REFINE: REFINE is a workbench which aids, in understanding the code structure of the COBOL programs, and in generating the documents which facilitate the understanding and maintenance of the system.

1.3   The Thesis

The organization of the report is as follows

- **Chapter 2** A brief description about file handling facilities in COBOL and data management facilities in RDBMS.

- **Chapter 3** Introduction to reverse engineering and definition of related terms.

- **Chapter 4** Description of SQLC, which is the tool for re-engineering information systems developed in COBOL to RDBMS. Also description of how the data model is extracted from the existing systems and how it is converted to the relational model.

- **Chapter 5** Details about tools in SQLC.

- **Chapter 6** Description of SQLC's graphical user interface.

- **Chapter 7** Discusses on testing the Data model recovery tool.

- **Chapter 8** Conclusions and scope for future work.

Chapter 2

Introduction to COBOL and RDBMS

This chapter gives brief introduction to COBOL and Relational Databases.

2.1 COBOL

In this section the main features of ANSI COBOL which are relevant to the present work are discussed [PK78].

2.1.1 Structure of COBOL Programs

Every COBOL program consists of four divisions

Identification division

Environment division

Data division

Procedural Division

The *Identification division* specifies the information required to identify the program by any one who may use it in the future. The main use of this division is for documentation.

In the *Environment division* the environment in which the program is to be run is defined, i.e. it specifies the platform and all the peripherals required by the program for its compilation and execution. This will have two sections. One is the configuration section and the second section is the input-output section. In the input-output section names are given to each of the data files used by the program. In this section the access mode, organization and keys for the files are also specified.

In the *Data division* all the data items used in the program and the structure of the records in each file are defined. The data division is divided into file, working storage, linkage and report sections. The first section is the file section which will have the details of every file used for input and output. The file section is the most important section for capturing the data model. In this section, a COBOL name is given to the temporary storage area for each file. The temporary storage area will be used to hold each record in turn as it is read in. This COBOL name is called the record name of the corresponding file. Each record name is followed by its structure.

The *Procedural Division* is the part of the COBOL program which is concerned with the actual operations to be performed on the data which have been defined in data division.

2.1.2   Record Descriptions in COBOL

The format of the COBOL record is hierarchical in structure and this is represented with the help of level numbers. These level numbers are in the range 1-49, the level number one being the most inclusive. The COBOL records consist of group fields and elementary fields. The elementary field cannot be broken down further and the group field is further broken into elementary or group fields. The subfields of a group field will have their level numbers higher then the level number of the group field. The size and type of elementary fields are defined by a picture clause specified with the field. The size of a group field is the sum of sizes of its elementary fields.

6

■ *The PICTURE clause*

The PICTURE clause must be present in the description of every elementary field in the data division, except in a few cases where the SIZE and CLASS clauses are used. The purpose of the PICTURE clause is to describe the number of characters in the data field and their type.

example:
    02 NAME PIC X(5).
    It specifies the NAME is of length 5 characters and the type of
    the NAME is alphanumeric.

Elementary field types in COBOL may be Numeric, Alphanumeric, Numeric edited, Alphanumeric edited or Alphabetic depending on the picture clause description of the field.

Example of record description in COBOL
    01 EMPLOYEE
        02 EMPLOYEE-NAME PIC A(25).
        02 EMPLOYEE-CODE PIC X(7)
        02 EMPLOYEE-ADDRESS
            03 STREET PIC X(30)
            03 CITY PIC X(30)
            03 PIN PIC 9(6)
        02 BASIC PIC 9(5)V99

In the example EMPLOYEE-NAME, EMPLOYEE-CODE, STREET, CITY, PIN and BASIC are elementary fields. EMPLOYEE-ADDRESS is a group field.

In the above example EMPLOYEE-NAME is of type alphabetic, EMPLOYEE-CODE is of type alphanumeric, PIN is of type numeric. BASIC is of type numeric with size 7 and 2 digits after decimal point.

2.1.3 File Handling in COBOL

The data which a COBOL program reads and writes is organized into files, which are made up of individual records. The organization of these records in the file may be sequential, relative or indexed [PK78].

In the *Sequential file organization* records are stored in the same order that they are written. The accessing mode for these files is sequential, i.e. the records can be accessed only in the order in which they are stored.

In the *Relative file organization* records are identified by a relative record number which specifies the position of the record from the beginning of the file. The access mode allowed is either sequential or random. In the first case the records are accessed in the order in which they are stored and in the second case the programmer accesses the record by specifying the relative record number.

In the *Indexed file organization*, the file is organized as an indexed sequential file on its primary key field. The records are stored either in ascending or descending order of the key field. These can be accessed either sequentially or randomly. In random access mode the value of the key field should be specified.

The important file operations in COBOL are

- *open* to open a file for reading or writing or appending the data.

- *close* to close the file.

- *write* to write a new record in to the file.

- *rewrite* to write the modified record in the file.

- *delete* to delete a record from the file.

2.2 RDBMS

2.2.1 Introduction

A Relational database represents the data and the relationships among data by a collection of tables, each table having a number of columns with unique names. A row in a table represents a relationship among a set of values. The data from a relational database is accessed using a query language [EN94].

2.2.2 SQL

SQL is a query language with which data can be accessed from a database. SQL is the most widely used query language because of its English like syntax. SQL provides a full range of retrieval functions as well as update functions. SQL can be used as a "stand-alone" query language, in which the user simply expresses SQL queries and database updates and has these operations executed directly. It is also used in the form of an 'embedded SQL' from application programs written in high level languages like C, COBOL, PL/I etc.. SQL embedded in C is more popular than others.

2.2.3 C with embedded SQL

A 'C' program which accesses a relational database through SQL statements embedded in it is called a *C with embedded SQL* program. To access data through such a program, data should be passed between the relational database and the program. This involves defining variables, cursors and arrays, integrating these variables into SQL commands, and converting data between the host language and the RDBMS.

Any variable that is inserted into a SQL statement in the program has to be declared in the DECLARE section and is called a *host variable*.

9

Cursor is a pointer to a tuple in the result of a query. Cursor is used in allowing tuple-at-a-time processing by *C host language*. The cursor is defined with the SQL query command in the program.

The DECLARE section is placed before main() and it is framed by commands as shown below:

EXEC SQL BEGIN DECLARE;
variable declarations
EXEC SQL END DECLARE;

The SQL commands embedded in C programs will have the prefix "EXEC SQL". These statements can be used to Insert, Delete and Update the records in a database.

Chapter 3

Software Re-engineering & Objectives

3.1   Introduction

The undeniable reality of software systems development is that year after year a lions share of the effort goes into modifying and extending existing systems about which we know very little. Re-engineering these systems is the most practical way of maintaining them. In this context the objective of re-engineering is to gain sufficient design level understanding of the system and utilising this information to enhance and to maintain the system. In this chapter the terms Reverse Engineering, Restructuring, and Re-engineering are defined.

3.2   Reverse Engineering

Reverse Engineering is the process of analyzing a subject system in order to identify the system's components and their inter-relationships and to create a representation of the system possibly at a higher level of abstraction [CCMJ92].

Reverse Engineering encompasses a wide array of tasks related to the understanding of software systems. Among these tasks are identifying the components

of an existing system and the relationships among these components, and creating high level descriptions of various aspects of the existing systems.

Re-documentation and design recovery are two major subfields of reverse engineering. The terms re-documentation and design recovery can be defined as:

Re-documentation: It is the creation or revision of the representations of a subject system that are intended to reflect certain software related characteristics inherent to the system.

Re-documentation provides alternate views of the system to help the users understand it. Displaying code listing in an improved form like goto-less form, and generating control flow diagrams directly reflecting the code are some examples of re-documentation.

Design Recovery: Design recovery recreates design abstractions from a combination of code, existing design documents, personal experience, and general knowledge about the problem and application domain.

3.3 Restructuring

Restructuring is the transformation of a software system from one representation to another, usually at the same relative abstraction level, while preserving the subject system's functionality and semantics [CCMJ92].

Some of the common restructuring tasks are:

- *code-to-code* transformations that recast a program from an unstructured form to a structured or goto-less form.

- *data-to-data* restructuring transformation to improve the logical model for the database design process.

Restructuring generally involves some form of reverse engineering from the original representation to some intermediate form. This is followed by altering this intermediate form, without changing its functionality or external behavior to get the same level of abstraction as of the system that is being restructured.

Restructuring can some times be performed only with the knowledge of structural form, without any knowledge about the program functionality. An example of this is converting a series of *if* statements to a *case* statement.

3.4   Re-engineering

Re-engineering is the examination and alteration of a subject system to reconstitute it in a new form, and the subsequent implementation of the new form. Implied in this term is the possibility of change in the essential requirements rather then a mere change in the form [CCMJ92].

Re-engineering involves both reverse engineering and forward engineering. In re-engineering, reverse engineering is used to achieve a higher level abstract description of the system. The abstract description is modified to incorporate new requirements of the system. This is followed by forward engineering or some restructuring.

The main purpose of reverse engineering is to understand the subject system. This is achieved through examining the subject system and representing it in a form which is more understandable to human-beings and which is less implementation dependent.

There are different levels of understanding, depending on the components of the subject system that are to be understood. Re-documentation and design recovery are two subsets of reverse engineering. Re-documentation facilitates understanding at the implementation and structure levels whereas design recovery facilitates understanding at the functional and domain levels.

3.5 Objectives

The purpose of reverse engineering a software system is to increase the overall understanding of the system that facilitates maintenance, reuse and overall quality assurance.

- Reverse engineering aids the *maintenance* process by providing alternate views of the system, detecting side effects and recovering lost documents.

- Reverse engineering methods are applied to find the *reusable* components in the existing system.

- The various artifacts that are used in *software quality assurance* can be derived by reverse engineering.

Chapter 4

SQLC: Migration from COBOL to RDBMS

4.1 SQLC

SQLC is a graphical workbench for migrating COBOL programs to a relational platform. It provides tools which aid in restructuring COBOL programs to 'C' programs which access relational database through the SQL statements embedded in them. These tools are discussed in detail in the next chapter.

The re-engineering of COBOL programs can be viewed as two parts. The first is re-engineering the data model and the second is restructuring the application programs.

4.2 Phases in SQLC

There are three main phases in SQLC. They are

(1) parsing

(2) unification

(3) conversion

In the parsing phase, we build the view of the data model as seen by different programs. This involves flattening the record structure and naming the fields in the record to eliminate name clashes between fields.

In the unification phase, we try to reconcile all the views of the data model found in the parsing phase to get a single integrated view of the data model. This involves finding the temporary files, i.e. files that are not part of database, generating the relational tables and providing the information required for the next phase.

In the conversion phase, the application programs are restructured. This involves defining host variables, replacing file access statements with SQL statements and restructuring other COBOL statements to 'C'.

In the parsing and unification phase the data model is re-engineered and in the conversion phase the application programs are restructured.

4.3   Re-engineering Data Model

Re-engineering the data model is the first step in re-engineering COBOL programs to 'C' embedded SQL, since the whole transformation depends on the recovered data model. This section discusses issues in data model recovery and describes the data model recovery tool in SQLC.

The process of data model recovery from the COBOL programs is based on the following facts:

- For each file in the data model, the external file name is unique to all the programs using it.

- In the data file, only the elementary fields at the leaves of the record description are represented as data.

- The data accessed by any program from a given data file will be the same, even though the record structure may be different in different programs. Therefore, the order of the fields in different record structures must be the same.

4.3.1 Issues in re-engineering the data model

The existing system developed in COBOL will not have a single integrated view of the data model. So, a better approach in re-engineering the data model is to get the view of the data model as seen by each program and then unifying all these views to get a single integrated view of the data model.

The problems in recovering the data model that arise due to data declarations are listed below.

- In COBOL programs it is possible that the same item may have different names in different programs or different items may have same name in different programs. This may be because the programmers do not follow uniform naming standards.

- In COBOL it is possible that the declaration of a data item does not give its full structure.

- If only some divisions of a group item are explicitly used in a program then the programmer may declare other subdivisions as FILLER. In this case the complete structure of the data item is not known.

- If the programmer is not interested in the subdivisions of a group item, he may declare the whole item as an elementary field. In this case also the structure of the group item is not known.

- It is possible that the same data item can have different structures in different programs.

- The same data item can have different types in different programs.

17

- In COBOL, the data item name can have the '-' character, which 'C' does not permit. This can be replaced with the character '_'.

Examples which illustrate the above conflicts in COBOL programs:

Student record in program A:

```
01 STUDENT
    02 NAME PIC X(25)
    02 ROLL-NO PIC 9(7)
    02 PROGRAM PIC X(12)
    02 DEPARTMENT PIC X(4)
    02 ADDRESS
        03 ROOM-NO PIC XX999
        03 HOSTEL PIC X(4)
```

Student record in program B:

```
01 STUDENT
    02 STUDENT-NAME PIC X(25)
    02 ROLL-NO PIC X(7)
    02 FILLER PIC X(16)
    02 ADDRESS PIC X(9)
```

Student record used in program C:

```
01 STUDENT PIC X(47)
```

In the above examples the DEPARTMENT and PROGRAM fields are not explicitly needed in program B. So, these two fields are clubbed together and referred to as FILLER. In a group item more then one item can be referred to using FILLER.

In program B, NAME is referred to as STUDENT-NAME, i.e. the same data item is referred to with different names in the two programs.

In program A, ROLL-NO is of type numeric, where as in program B, it is of type alphabetic.

In program C the whole record is considered as a single field.

Apart from the conflicts in the data declarations, one more thing to keep in mind is the existence of temporary files that are used to hold data temporarily, files used for sorting transactions, etc.. For these files, the view of the data model as seen by different programs may not be unifyable because these files are not part of the data base. Some temporary files can be recognised by their declarations, for example, all sort files are temporary. If a file is used only once throughout the application programs, then that file can be treated as a temporary file. However, for these type of files the user is the sole authority to decide whether they are temporary or not.

It is possible to extract some semantic information about the domain of each data item and the functional dependencies, by examining the statements in the procedure division. This will help in defining the types in the relational database system, and in enforcing the domain integrity constraints.

4.3.2 Parser or Relations extractor

This is the first phase in re-engineering COBOL programs. In this phase we analyse the INPUT-OUTPUT SECTION in the Environment division and the FILE SECTION in the DATA DIVISION.

The INPUT-OUTPUT section is analysed to find out the physical files that are used in the program, and the FILE-SECTION is analysed to find the view of the file as perceived by the program. In this phase all the record descriptions are flattened.

In the flattened record description the following symbols are used to represent the type of the field and to indicate whether it is a key field or not.

    s : Character string
    i : Integer
    f : Float
    k : Key field
and n : Not a key field.

For each data file in the data base, the list of programs using it and the corresponding views are found.

Example:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT CRS-FILE ASSIGN TO "COURSE"
    ORGANIZATION INDEXED
    ACCESS MODE RANDOM
    RECORD KEY IS CRS-ID.
SELECT STUD-FILE ASSIGN TO "STUDENT".
DATA DIVISION.
FILE SECTION.
FD CRS-FILE.
01 CRS-REC.
    02 CRS-ID PIC X(6).
    02 CRS-NAME PIC X(40).
    02 INSTR PIC X(25).
    02 UNITS PIC 9.
FD STUD-FILE.
01 STUD-REC.
    02 ROLL-NO PIC 9(7).
    02 STUD-NAME PIC X(25).
    02 PRESENT-ADDR
        03 ROOM-NO PIC X(5).
        03 HOSTEL PIC X(25).
    02 PERMANENT-ADDR
        03 HOUSE-NO PIC X(10).
        03 STREET PIC X(25).
        03 CITY PIC X(25).
        03 PIN PIC 9(6).
```

```
        02 UNITS-DONE PIC 99.
        02 CPI PIC 9V99.
```

In the above example, COURSE and STUDENT are two data files used in the program and the corresponding logical files are CRS-FILE and STUD-FILE respectively. The record description of CRS-FILE is CRS-REC and that of STUD-FILE is STUD-REC. COURSE file is organised as an indexed sequential file with its primary key as CRS-ID.

The flattened record corresponding to CRS-REC is

```
        crs.rec.crs_id s 6 k
        crs.rec.crs_name s 40 n
        crs.rec.instr s 25 n
        crs.rec.units i 1 n
```

The flattened record corresponding to STUD-REC is

```
        stud.rec.roll_no i 7 n
        stud.rec.stud_name s 25 n
        stud.rec.present.addr.room_no s 5 n
        stud.rec.present.addr.hostel s 25 n
        stud.rec.permanent.addr.house_no s 10 n
        stud.rec.permanent.addr.street s 25 n
        stud.rec.permanent.addr.city s 25 n
        stud.rec.permanent.addr.pin i 6 n
        stud.rec.units_done i 2 n
        stud.rec.cpi f 3.2
```

If OCCURS clause is present in the record description, i.e. in case of arrays, then the flattened record is obtained by duplicating the field as many times the array size. If a group field is of array type then its sub-record structure is duplicated as many times the array size.

Example:

```
01 STUDENT.
    02 ROLL-NO PICTURE 9(7).
    02 NAME PICTURE X(25).
    02 YEAR PICTURE 9(4).
    02 SEMESTER PICTURE 9.
    02 COURSE OCCURS 2 TIMES.
        03 NUMBER PICTURE X(5).
        03 WEIGHT PICTURE 9.
        03 GRADE PICTURE X.
```

In the above record description COURSE is an array of length 2, and having three sub fields in it.

The flattened record structure for the above example is:

```
student_roll_no i 7 n
student_name s 25 n
student_year i 4 n
student_semester i 1 n
student_course1_number s 5 n
student_course1_weight i 1 n
student_course1_grade s 1 n
student_course2_number s 5 n
student_course2_weight i 1 n
student_course2_grade s 1 n
```

4.3.3 Unification

Unification is the process of reconciling the different views of the data model as seen by different programs to get a single integrated view. This process takes as its input, flattened record structures obtained in parsing phase. These records are unified to get a single record. This record is converted into a table in the relational database.

Two record descriptions are identified as being of the same record by means of the external file name which stores these records. The following rules are applied in unifying the fields in the data model.

Here A is the field in the first record and B is the field in the second record.

- A and B are unifyable if they are of the same size and are of the same type. The resulting field will have the same type and size as of A or B.

- If A & B are of the same size and one of the fields is of filler type then they are unifyable. The resulting field will have the same type as that of the other field.

- If field A is of type alphabetic and field B is of type numeric, and the size of field A is equal to size of field B then A & B are unifyable. The resulting field is of the same type as that of B.

    If B is of type alphabetic and A is numeric, then the resulting field is of the same type as A.

- If the size of field A is greater than the size of field B, and A is of type alphabetic or filler, then A is broken into fields A1 & A2. The size of A1 is equal to that of B and size of A2 is equal to the difference in the sizes of A & B. A1 and B are unified to get a field having the same size and type as of B. A2 is then unified with the next field of the second record.

Similarly if size of field B is greater than that of A, and B is of type alphabetic or filler, then B is broken into two fields B1 and B2. A is unified with B1 and B2 is unified with the next field of the first record.

Two records are unifyable if all the fields in them are unifyable. Here is an example that illustrates the unification process.

Example:
        View 1
        employee_name s 25 n
        employee_number s 10 n
        employee_designation s 4 n
        employee_basic f 8.2 n
        employee_address s 80 n

        View 2
        employee_firstname s 15 n
        employee_lastname s 10 n
        employee_code s 10 n
        filler1 z 12 n
        employee_addr_houseno s 10 n
        employee_addr_street s 25 n
        employee_addr_city s 25 n
        employee_addr_pin i 6 n

In this example, View 1 and View 2 represent two different views of an employee record as seen by two different programs. To unify these two views, proceed from left to right in both the cases.

Look at employee_name and employee_firstname. Here, the sizes are different. So, the employee_name is broken into two fields employee_name1 of size 15 and

24

employee_name2 of size 10 (15 - 10). The field employee_name1 unifies with employee_firstname resulting in a field employee_firstname of size 15 and type string, and the algorithm proceeds. Here, one comes up with employee_name2 and employee_lastname. These two fields are of the same type and of same length. Hence, these two fields are unifyable and the resulting field is employee_lastname of size 10 and type string.

Next, proceed to employee_number and employee_code. These two fields are of the same type and of the same size. So, these two fields are unified to get employee_number of type string and length 10.

The next fields to be unified are employee_designation and filler1. The sizes of two fields are different. So, filler1 is divided into filler1_1 of length 4 and filler1_2 of length 8. filler1_1 is unified with employee_designation resulting in employee_designation, and filler1_2 is unified with employee_basic resulting employee_basic of type float and the algorithm proceeds.

The current fields in the unification process are employee_address and employee_addr_houseno. Here employee_address is broken into two fields employee_address1 and employee_address2. The field employee_address1 is unified with employee_houseno resulting in employee_houseno. Again employee_address2 is broken into two fields employee_address2_1 and employee_address2_2. The field employee_address2_1 is unified with employee_addr_street resulting in employee_addr_street. The field employee_address2_2 is again broken into two fields. The first field is unified with employee_addr_city and second field is unified with employee_addr_pin.

Now, all the fields in the two views are exhausted. So, the unification process terminates resulting the unified view given below.

Unified view of the employee record:

employee_firstname s 15 n

employee_lastname s 10 n

employee_number s 10 n

employee_designation s 4 n

employee_basic f 8.2 n

employee_addr_houseno s 10 n

employee_addr_street s 25 n

employee_addr_city s 25 n

employee_addr_pin n 6 n

4.3.4 Design of Relations

Designing relations from the unified record description is a simple and straight forward procedure. For each file in the data model an equivalent relation is defined using DDL statements. The fields in the integrated view of the record description of the file are made as the fields in the relation corresponding to the file. The relation table definition corresponding to employee file whose integrated view of the record was given above is

create table employee
```
    (
            employee_firstname char(16),
            employee_lastname char(11),
            employee_number char(11),
            employee_designation char(5),
            employee_basic float,
            employee_addr_houseno char(11),
            employee_addr_street char(26),
            employee_addr_city char(26),
            employee_addr_pin integer
    )
```

4.4 Restructuring the application programs

Restructuring COBOL programs to C embedded SQL involves declaring *host variables*, defining *cursors*, defining equivalent variables in C for the variables present in the WORKING-STORAGE section of the COBOL program, changing file access statements to their equivalent SQL statements and changing other statements in the procedure division to their equivalent C statements.

In the file system, data transfer between the file and the program logic is done using the record area given in the file description entry, where as in SQL it is done through host variables. So, for each file a set of host variables representing the records of the file are defined.

The cursor is a pointer to individual rows in the SQL query result. Each data file (converted to a relation) accessed in the program will have a cursor field defined. This is mainly useful in sequential accessing of the data from the relational tables.

The file access statements like OPEN, CLOSE, READ, WRITE, REWRITE and DELETE of COBOL are replaced with corresponding *open, close, fetch or select, insert into, update* and *delete* statements of SQL with simple queries.

The other statements in the procedure division are restructured to C statements using restructuring rules given by A. Satish Kumar [SK96].

Chapter 5

SQLC: Tools

The SQLC provides the following tools.

- parser

- unifier

- relationer

- slicer

- converter

Each of the above tools is discussed in following heads.

5.1 Parser

The Parser is a grammar based tool which analyses the Data division statements in the COBOL program to get the data view as seen by the program. The Parser also generates the list of subroutines called in the program. This information is used in by the Slicer tool in displaying the PDG.

The Parser flattens the COBOL record to avoid name clashes between the fields in the record. The flattening of records is done because the table structure in relational
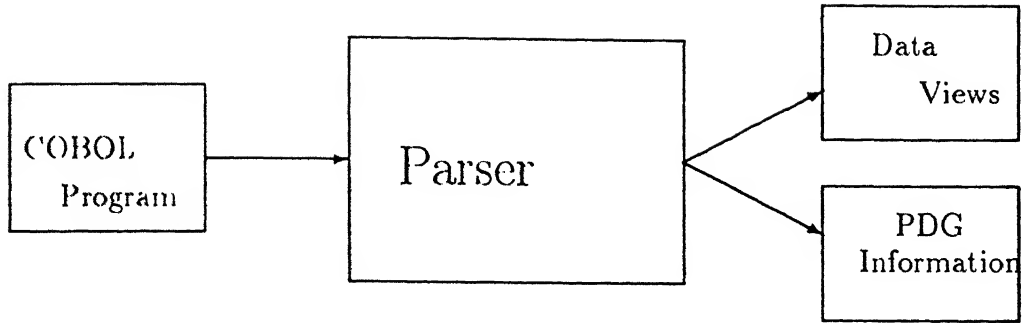
Figure 5.1: Parser

database is flat, and the record structure in COBOL is hierarchical. All the items in the flattened record will be elementary items. The names of the fields in these records are formed by concatenating names of the all top level data items corresponding to record field. These flattened record structures are called file description entries.

The file description entry will represent the view of the corresponding data file as seen by the program. For each data file in the database we build the list of programs using the data file and the corresponding file description entries. These file description entries are used by unifier to get integrated view of the record.

5.2 Unifier

The Unifier is a tool for integrating different data views of a file in the existing system's database, as seen by different programs. The Unifier tool takes file description entries generated by Parser as input. For each file in the data base, all the file description entries corresponding to the file are unified to get an integrated view of the file. The integrated view contains the basic record structure of the data file. The relational table corresponding to the data file is formed with the fields present in the integrated view.

The Unifier generates the DDL statements for defining tables. It also generates the corresponding information between the record fields of the file and the fields of

29
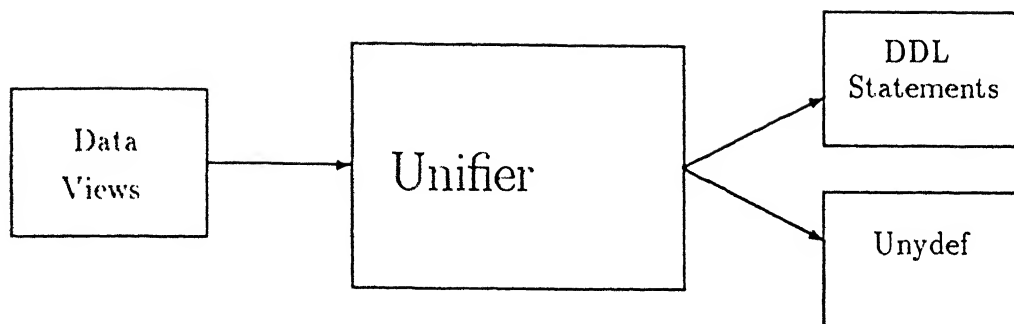
Figure 5.2: Unifier

the relational table in a intermediate form named Unydef. Unydef notations are given in appendix A.

5.3 Relationer

Relationer is a tool used

- to display all the relational tables in the recovered database,

- to interact with the user in finding the temporary files,

- and to change the names of the fields in the relational tables and incorporate these changes in Unydef.

Relationer lists all the tables involved in the database. A user who has knowledge of the system can specify the relations which are in the recovered model and are not part of the database as temporary files. An example is the table corresponding to the report file in the program is not part of the database.

As the field names in the tables are selected randomly while unifying, they may not me meaningful. This tool provides the environment in which you can change the field names and incorporate these changes in the information that is passed to converter.
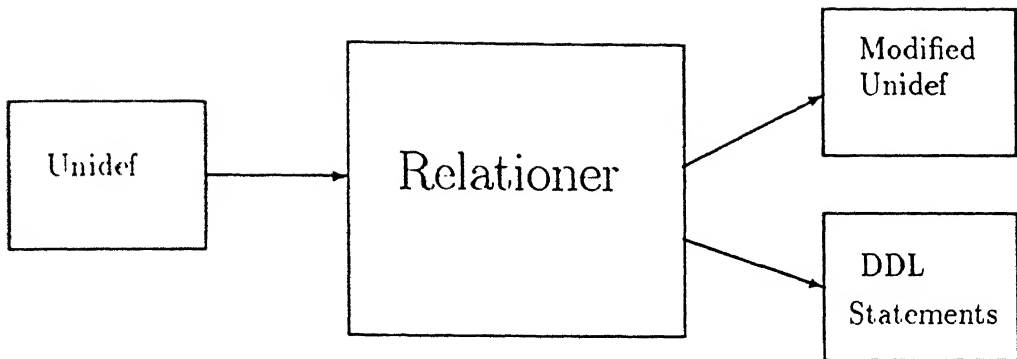
Figure 5.3: Relationer
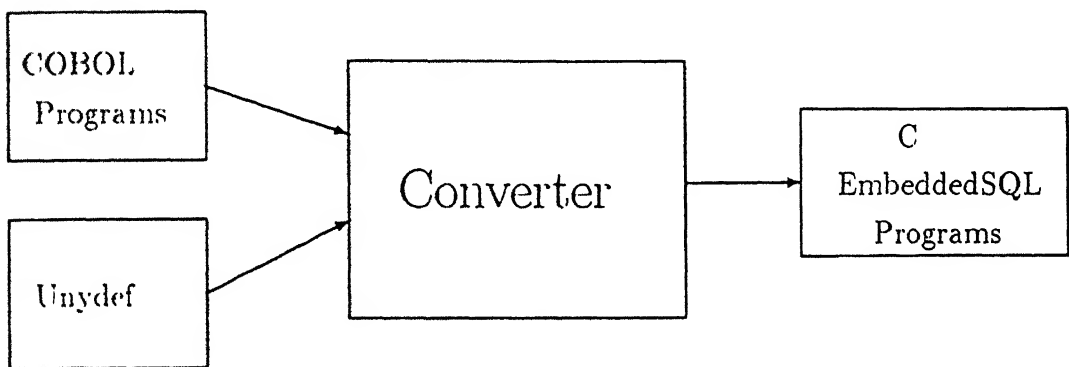
Figure 5.4: Converter

5.4 Converter

The Converter is a grammar based tool developed by Satish Kumar, A [SK96]. for transformation of COBOL programs to C-SQL programs. The transformation is performed by applying restructuring rules. The converter takes the recovered data model represented in Unydef and COBOL program as input. The Converter generates C-SQL program.

5.5   Slicer

Slicer is a tool which constructs and Displays Program Dependence graph of the system. The input for the slicer is the dependence information produced in the parsing phase. The main use of PDG is if user is interested in re-engineering only some programs than it helps user to find out what all programs to be re-engineered. It can also be used in identifying the reusable components of the subject system. The transitive closure of the node corresponding to program A gives the list of the programs on which A is dependent.

Structure of PDG: The nodes of PDG corresponds to programs of the subject system. There will be an edge between node corresponding to program **A** to node corresponding to program **B**, if program **A** calls program **B**.

Dependencies: The various programs on which a given program **A** is dependent can be find by generating the transitive closure of the node corresponding to program **A**.

Chapter 6

SQLC: User Interface

A user interface which runs on X-windows has been implemented in Motif for providing better interaction for user. This chapter explains the functionalities of user interface and describes the usage of interface in the process of Migration.

6.1 Main Window

Figure 6.1 shows the *main window* of the SQLC tool. The main window has

- *Menu bar*, used to select different tools available in the SQLC.

- *Message window*, used to display the messages.

As seen in Figure 6.1, the menu bar consists of eight menu buttons namely - *File, Parser, Unifier, Slicer, Relationer, Converter, Quit* and *Help.*

The File button when selected will create a window which is used to select COBOL files.

The Parser button when selected invokes the tool *Parser.*

File   Parse   Unify   Slicer   Relationer   Converter   Quit   Help

Messages:

/2d1/ssk/thesis/project/cobprogs/set11/dispdcha.cbl
/2d1/ssk/thesis/project/cobprogs/set11/set111/addbudget.cbl
/2d1/ssk/thesis/project/cobprogs/set11/set111/caddbudget.cbl
/2d1/ssk/thesis/project/cobprogs/set11/set111/subbgt.cbl
/2d1/ssk/thesis/project/cobprogs/set11/set111/addbudget.cbl
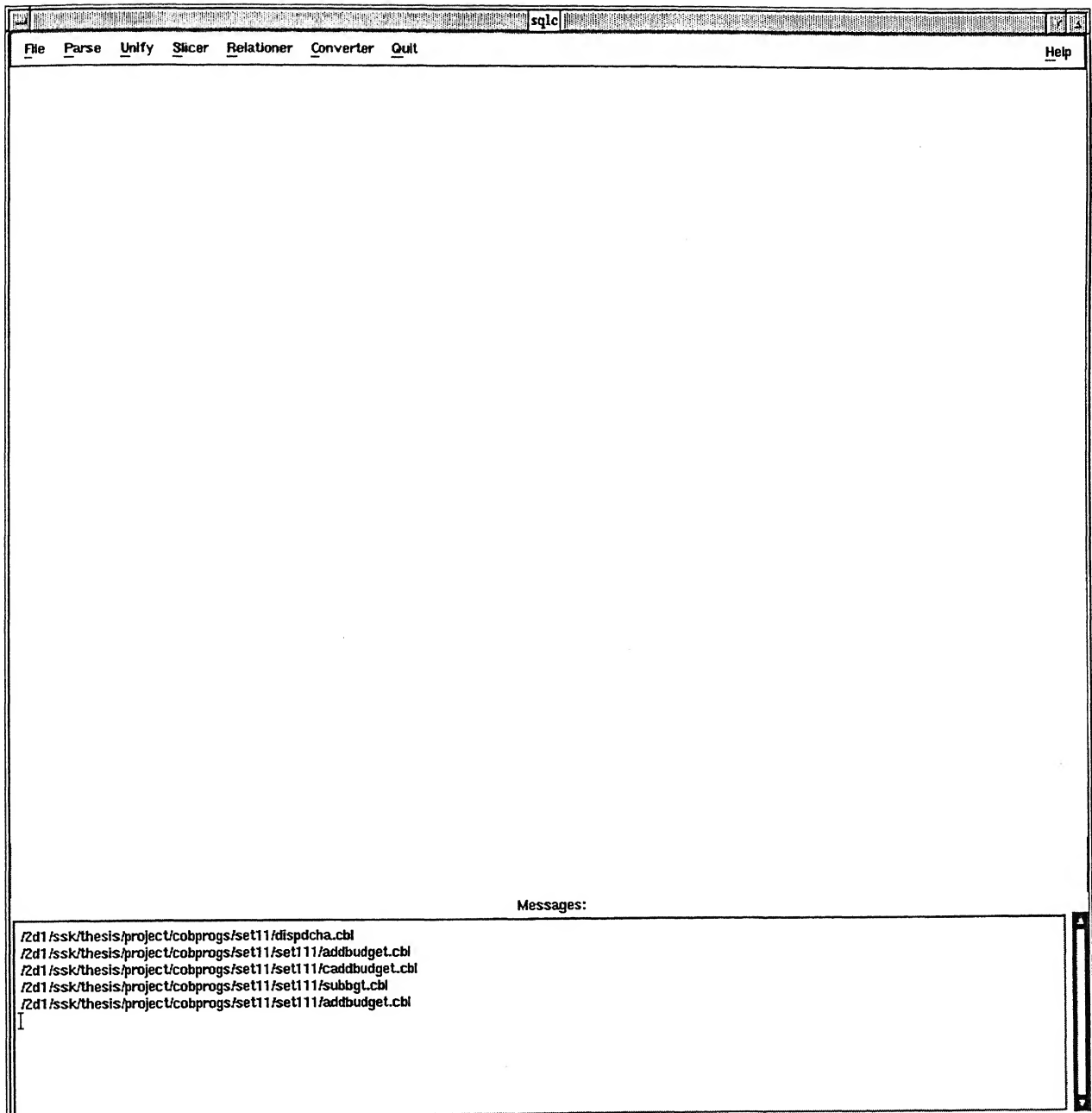
Figure 6.1 The Main window.

The Unifier button when selected invokes the tool *Unifier*.

The Slicer button when selected invokes the tool *Slicer*.

The Relationer button when selected invokes the tool *Relationer*.

The Converter button when selected invokes the tool *Converter*.

The Quit button when selected will terminates the SQLC tool.

The Help button when selected will display help information in pop-up window.

6.2   File Selection Window

File selection window is shown in figure 6.2., is used to select a set of COBOL programs which have to be converted to relational platform. The function of each button is given below.

- *OK* button is used to add the highlighted file to the selected file list.

- *Remove* button is used to remove the marked file name from the selected file list.

- *Clear* button is used to remove all the entries from the selected file list.

- *Cancel* button is used to end the File selection session.

```
┌──────────────────────────────────────────sqlc──────────────────────────────────────────┐
│                                                                                          │
│         Relation names :                              Field names :                      │
│  ┌──────────────────────────┐  ┌──────────────────────────────────────────────────────┐ │
│  │ card_recs                │  │ card_std_name                                        │ │
│  │                          │  │ card_street                                          │ │
│  │                          │  │ card_addr                                            │ │
│  │                          │  │                                                      │ │
│  │                          │  │                                                      │ │
│  │                          │  │                                                      │ │
│  │                          │  │                                                      │ │
│  │                          │  └──────────────────────────────────────────────────────┘ │
│  │                          │                      Change :                              │
│  │                          │  ┌──────────────────────────────────────────────────────┐ │
│  └──────────────────────────┘  │ I                                                    │ │
│                                 └──────────────────────────────────────────────────────┘ │
│                                                       To :                                │
│                                 ┌──────────────────────────────────────────────────────┐ │
│                                 │ I                                                    │ │
│                                 └──────────────────────────────────────────────────────┘ │
│                                                                                          │
│       ┌────────┐        ┌────────┐        ┌────────┐        ┌────────┐                    │
│       │  Save  │        │   Ok   │        │ Cancel │        │  File  │                    │
│       └────────┘        └────────┘        └────────┘        └────────┘                    │
│                                                                                          │
│                                                                                          │
│                                         Messages:                                        │
│  ┌───────────────────────────────────────────────────────────────────────────────────┐ │
│  │ I                                                                                   │ │
│  │                                                                                     │ │
│  │                                                                                     │ │
│  │                                                                                     │ │
│  └───────────────────────────────────────────────────────────────────────────────────┘ │
└──────────────────────────────────────────────────────────────────────────────────────────┘
```

Figure 6.2 The File Selection window.

6.3 Relationer Window

The relationer window is shown in the figure 6.3. The relationer window provides interface for changing the field names in the table and to interact with user to find the temporary file present in the system. The relationer window consists of:

- Two List widgets named

    - *Relation names*, displays all the relations present in the recovered data model.

    Field names, displays all the fields present in the relation which is highlighted in the list *Relation names*.

- Two text widgets named

    - *Change*, contains the name of the field to be changed.

    - *To*, takes the new name to be given to the field present in the *Change* widget.

- Four Button widgets named

    File, when pressed will assume that the relation highlighted in the *Relation names* is not part of the data model. It considers it as a temporary file while converting the COBOL program to C-SQL.

    - *Save*, when pressed saves the changes performed on the relations in Unydef.

    - *Ok*, when pressed save the changes and returns to main window.

    - *Cancel*, when pressed cancels all the changes performed on the relations.

Filter

sk/thesis/project/cobprogs/set11/set111/*

Selected Files:

/2d1/ssk/thesis/project/cobprogs/set11/dispdcha.cbl
/2d1/ssk/thesis/project/cobprogs/set11/set111/addbudget.cbl
/2d1/ssk/thesis/project/cobprogs/set11/set111/caddbudget.cbl
/2d1/ssk/thesis/project/cobprogs/set11/set111/subbgt.cbl

Directories

gs/set111/set1111
gs/set111/set1111/..

Files

add.cbl
addbudget.cbl
caddbudget.cbl
subbgt.cbl
temp

Selection

ject/cobprogs/set111/set1111/addbudget.cbl

| OK | Filter | Cancel | Help |

| Remove | Clear |

Messages:

/2d1/ssk/thesis/project/cobprogs/set11/dispdcha.cbl
/2d1/ssk/thesis/project/cobprogs/set11/set111/addbudget.cbl
/2d1/ssk/thesis/project/cobprogs/set11/set111/caddbudget.cbl
/2d1/ssk/thesis/project/cobprogs/set11/set111/subbgt.cbl
/2d1/ssk/thesis/project/cobprogs/set11/set111/addbudget.cbl

Figure 6.3 The Relationer window.

6.4 Converter Window

The converter window is shown in figure 6.4. The converter window provides interface in restructuring COBOL programs to C-SQL.

The converter window consists of:

- *Menu bar* which has menu buttons:

    - *File*, when selected will display a pop-up window which contains the list of COBOL programs that are being converted in the current session. If a file is selected from the list, it is displayed in the text widget *COBOL Program*.

    - *Options* is used to set the option for invoking converter tool on selective COBOL programs or on all the COBOL programs in present session.

    - *Converter* will invoke the converter tool on a selected program in case the option is set to selective. Otherwise it invokes the converter tool on all the COBOL programs in the file list. In case of selective option it displays the C-SQL program generated in the text window *SQLC program*.

    - *Save*, when selected the modifications done manually in the program generated by converter are saved.

        Main menu, when selected returns to the main window.

        Help, when selected will display help information in pop-up window.

- Two List widgets named

    - *COBOL program*, to display the COBOL program selected for conversion.

    - *SQLC program*, to display the generated C-SQL program by the converter.

File   Edit   Converter   Options   Main menu   Help

COBOL program:

```
IDENTIFICATION DIVISION
    PROGRAM ID PRNCARDS

ENVIRONMENT DIVISION.

INPUT OUTPUT SECTION

FILE CONTROL.
    SELECT CARD IN ASSIGN TO 'CARD_RECS'
    ORGANIZATION LINE SEQUENTIAL.

DATA DIVISION.
```

SQLC program:

```
# include <stdio.h>
# include <string.h>
# include "ex22.h"
# define END OF FILE (long)100
# define TRUE 1
# define FALSE 0
EXEC SQL include sqlca;
char cobSqlTemp[100];
char cobSqlTemp1[100];
    /* Data Division entries declarations */

        /* Host Variable declarations for file CARD IN*/
EXEC SQL begin declare section;
    char CARD ADDR[80 + 1];
EXEC SQL end declare section;

/* Working Storage entries declarations */

char END OF DATA INDICATOR[3 + 1];
```

Messages:

```
CONVERTING
ex22

]
```

6.5 Slicer
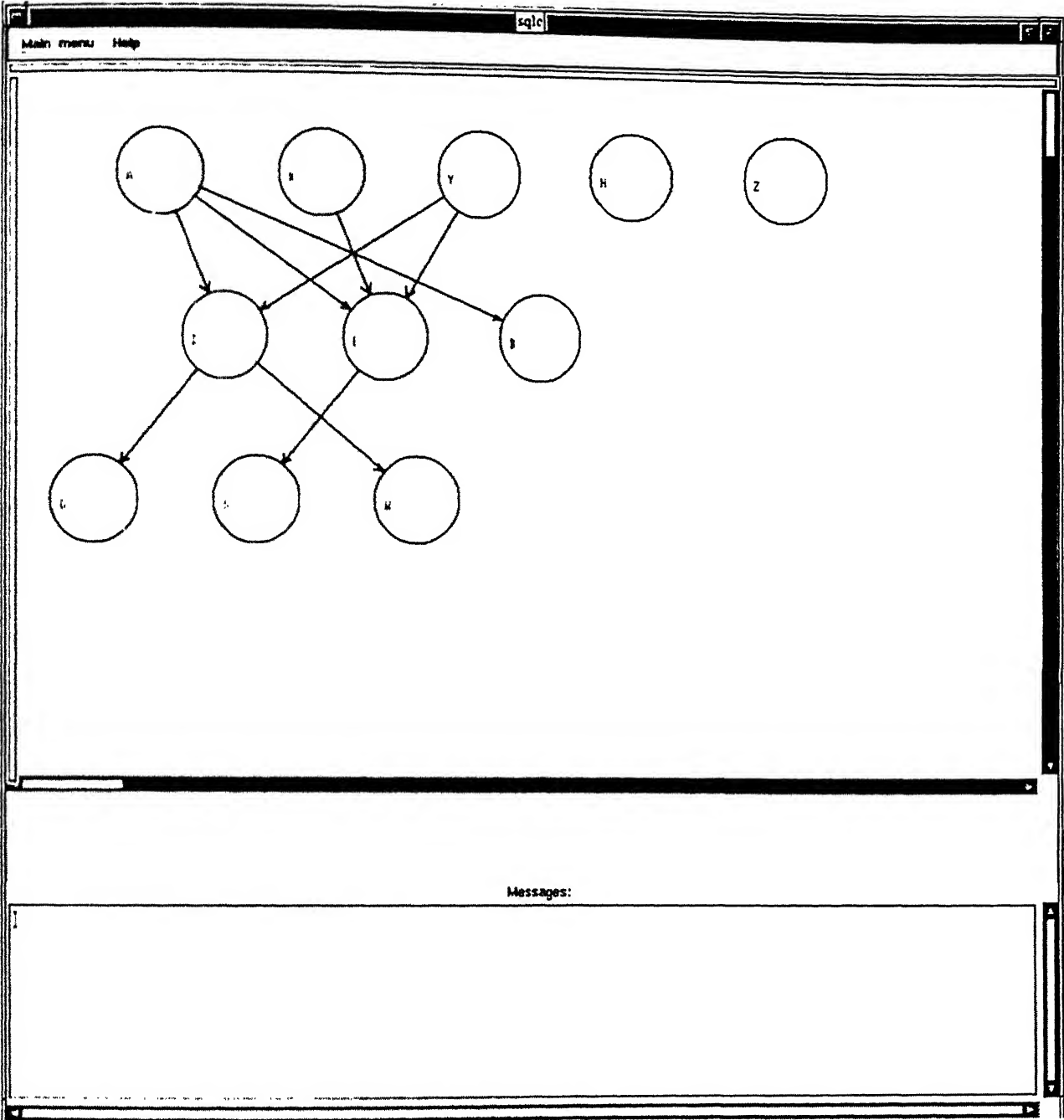

Figure 6.6 The Slicer window.

The Slicer window is used to display the Program Dependence Graph. It consists of

- *menu bar,*

- *drawing area.*

The PDG is displayed in the drawing area. A node in the graph represents the program. An edge from node A to node B states that program A calls program B. If there is a path from A to B, it indicates that the program A is dependent on program B.

Chapter 7

Testing & Discussion

This chapter outlines the testing procedure followed, to test the data model recovery tool, SQLC, in recovering the data model from a set COBOL programs. This chapter also discusses extraction of data model with examples.

7.1 Testing

We have tested our implementation by running several existing systems implemented in COBOL, and verifying the recovered data model with manually extracted data model of the system. In this section, we describe some of the systems implemented in COBOL, and evaluate the recovered data model of the system.

For every relational table present in the extracted data model, we compare the type and size of each field in the relational table with the corresponding field in the basic record description of the data file corresponding to the relation. The fields in the basic record description of the file are found manually, or taken from the person having knowledge of the existing system.

7.1.1 Test suite

■ *Test set #1*

The *test set #1* contains two COBOL programs having average length of 100 lines. These two programs use three different files to store the data. The three files are converted to three relational tables. The COBOL programs in this set, and extracted relations from these programs are given in Appendix C.

■ *Test set #2*

The *test set #2* contains four COBOL programs. Average length of each program is 80 lines. The programs in this set refer four files to store the data. Two of the three data files referred in the programs are converted to equivalent relational tables. The COBOL programs in this set, and the relational tables extracted are given in Appendix D.

■ *Test set #3*

This set contains 15 COBOL programs with an average length of 90 lines. These fifteen programs use 18 different files for maintaining data. Sixteen of the eighteen files are converted to relational tables.

■ *Test set #4*

This set contains 18 COBOL programs with an average length of 80 lines. These programs use 18 files to handle data. All the data files referred in these programs are converted to equivalent relational tables.

■ *Test set #5*

This set contains 40 COBOL programs. Average length of each program is 90 lines. These programs use 40 files for data maintenance. Thirty seven of these forty data files are converted to relational tables.

■ *Test set #6*

This set contains 40 COBOL programs having average length of 80 lines. These programs are referring to fifty files. Forty seven of these fifty files are converted to relational tables.

7.2 Examples

Example 1:-

File description of "user.dat" in Program **A** is:

```
      ...    ...
      FILE-CONTROL.
          SELECT USERS ASSIGN TO "user.dat".
      DATA DIVISION.
      FILE SECTION.
      FD USERS
          LABEL RECORDS ARE OMITTED
          DATA RECORD IS USER.
      01 USER.
          02 ID PICTURE IS X(8).
          02 NAME PICTURE IS X(25).
          02 FILLER PICTURE IS X(55).
```

File description of "user.dat" in Program **B** is:

```
      FILE-CONTROL.
          SELECT INPUT-FILE ASSIGN TO "users.dat".
      DATA DIVISION.
      FILE SECTION.
      FD INPUT-FILE LABEL RECORD OMITTED
          DATA RECORD USER-RECORD.
      01 USER-RECORD.
```

```
        02 INFORMATION PIC X(33).
        02 ADDRESS.
            03 STREET PICTURE X(25).
            03 CITY PICTURE X(30).
```

View of the "user.dat" as seen by program A is,

```
        user_id s 8 n,
        user_name s 25 n,
        filler1 / 55 n
```

View of the "user.dat" as seen by program B is,

```
        user_information s 33 u,
        user_address_street s 25 u n,
        user_address_city s 30 u n
```

Unified view of the "user.dat" is,

```
        user_id string 8,
        user_name string 25,
        user_address_street string 25,
        user_address_city string 30
```

Example 2 :-

File description of "budget.dat" in Program A is:

```
        ...    ...
        FILE-CONTROL.
            SELECT BUDGET ASSIGN TO "budget.dat".
        DATA DIVISION.
        FILE SECTION.
        FD BUDGET
```

```
            LABEL RECORDS ARE OMITTED
            DATA RECORD IS BUDGET.
     01 BUDGET.
            02 V0 PIC X(8).
            02 FILLER PIC X.
            02 V1 PIC 9(2).
```

File description of "budget.dat" in Program **B** is:

```
            FILE CONTROL.
            SELECT INPUT-FILE ASSIGN TO "budget.dat".
     DATA DIVISION.
     FILE SECTION.
     FD INPUT-FILE LABEL RECORD OMITTED
            DATA RECORD I-REC.
     01 I-REC.
            02 UNAME PIC X(8).
            02 FILLER PIC XX.
            02 UBGT PIC 9(7).
```

View of the "budget.dat" as seen by program A is,

```
            budget_v0 s 8 n,
            filler1 z 1 n.
            budget_v1 i 2 n
```

View of the "budget.dat" as seen by program B is,

```
            i_rec_uname s 8 u,
            filler1 z 2 n,
            i_rec_ubgt i 7 n
```

The length of the budget record used in program **A** is 11, where as it is 17 in program **B**. So, the data file "budget.dat" is not converted to a relational table.

Chapter 8

Conclusions

Our main objective has been to provide tools for re-engineering a subject system, whose database application programs are written in COBOL, to relational database environment.

In this thesis, we have designed and constructed a work bench SQLC, which provides environment for restructuring COBOL programs of the subject system to C embedded SQL programs.

The main features of SQLC are:

- It finds the data model of the subject system by examining the application programs in the subject system.

- It transfers recovered data model to relational platform.

- It provides interface to alter the names of the fields in relational model.

The limitations of Data model recovery tool in SQLC are:

- It does not handle variable length records.

- It does not provide facility to normalise the tables of the recovered data model.

Future Work

- Data model recovery tool can be made more effective by providing a way to handle variable length records.

- In the present work the array items are duplicated as many times as the array size. Instead of duplicating the array item, the array can be treated as another relational table.

- This tool can be extended to normalize the relations in the data model.

- Integrating SQLC in a CASE tool.

Appendix A

Unydef

Unydef is the intermediate form to represent unified view of the data model and the corresponding information between the record fields and the table fields.

The meanings of the various symbols used in Unydef are:

s: Field type is char.

i: Field type is integer.

f: Field type is float.

z: Field is filler.

k: Field is a Key.

n: Field is not a key.

c: Field name is changed.

u: Field name is Unchanged.

r: Record field is unified with more then one table field.

Each field in the data view is represented as,

fieldname type size key-indicator

The type may be s, i, f or z depending on the type of the field.

Key-indicator is k if the field is a key, else it is n.

Example:

If the data view of the file **person** as seen by program **A** is:

PERSON_NAME s 25 k,

PERSON_ADDRESS s 70 n,

and the unified view of the file **person** is:

NAME s 25 k,

HOUSE_NO s 10 n,

STREET s 30 n,

CITY s 30 n,

Then the Unydef for the file **person** in program **A** is:

PERSON_NAME s 25 c NAME,

PERSON_ADDRESS r {

HOUSE_NO s 10 u,

STREET s 30 u,

CITY s 30 u,

}

Appendix B

Glossary

COBOL   : Common Business Oriented Language.

COBSQL : Tool for converting COBOL programs to COBOL with embedded SQL programs.

DDL   : Data Definition Language.

DML   : Data Manipulation Language.

PDG   : Program Dependence Graph.

RDBMS : Relational Database Management System.

SQL   : Standard Query Language.

SQLC   : Tool for Converting COBOL programs to C with Embedded SQL programs.

Appendix C

Example Programs: Set 1

This appendix provides a set of two COBOL programs, and the recovered data model from these programs.

Program "report.cbl"

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UPDATE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CRS-FILE ASSIGN TO "COURSE"
    ORGANIZATION INDEXED
    ACCESS MODE RANDOM
    RECORD KEY IS CRS-ID.
    SELECT STUD-FILE ASSIGN TO "STUDENT".
    SELECT ENROL-FILE ASSIGN TO "ENROLL".
DATA DIVISION.
FILE SECTION.
FD  CRS-FILE.
01  CRS-REC.
    02 CRS-ID    PIC X(6).
```

.

```cobol
    02 CRS-NAME  PIC X(40).
    02 INSTR     PIC X(25).
    02 FILLER    PIC 9.
FD   STUD-FILE.
01   STUD-REC.
    02  ROLL-NO      PIC 9(7).
    02  STUD-NAME    PIC X(25).
    02  ADDR         PIC X(30).
    02  FILLER       PIC X(5).
WORKING-STORAGE SECTION.
77  WS-ROLL-NO       PIC 9(7).
77  E-O-F        PIC 9 VALUE 0.
88 STUD-FILE-END    VALUE 1.
88 CRS-FILE-END     VALUE 2.
77  FLAG         PIC 9.
88 OVER VALUE 1.
PROCEDURE DIVISION.
MAIN-PARA.
    OPEN INPUT STUD-FILE.
    OPEN INPUT CRS-FILE.
    PERFORM  PRINT-STUDENT-LIST
        UNTIL STUD-FILE-END.
    PERFORM  PRINT-COURSE-LIST
        UNTIL CRS-FILE-END.
    CLOSE STUD-FILE.
    CLOSE CRS-FILE.
    STOP RUN.
PRINT-STUDENT-LIST.
    PERFORM GET-STUDENT.
    IF NOT STUD-FILE-END
        MOVE 0 TO FLAG
```

```
            PERFORM PRINT-LIST.
PRINT-LIST.
    DISPLAY "ROOL NO:" ROLL-NO.
    DISPLAY "ADDRESS:" STUD-NAME, ADDR.
GET-STUDENT.
    READ STUD-FILE RECORD AT END
                    MOVE 1 TO E-O-F.
PRINT-COURSE-LIST.
    PERFORM GET-COURSE.
    IF NOT CRS-FILE-END
        MOVE O TO FLAG
        PERFORM SUB-PRINT-LIST.
SUB-PRINT-LIST.
    DISPLAY "ROOL NO:" ROLL-NO.
    DISPLAY "ADDRESS:" STUD-NAME, ADDR.
GET-COURSE.
    READ CRS-FILE RECORD AT END
                    MOVE 1 TO E-O-F.
```

Program "enrol.cbl"

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID. UPDATED.
    ENVIRONMENT DIVISION.
    INPUT-OUTPUT SECTION.
    FILE-CONTROL.
        SELECT CRS-FILE ASSIGN TO "COURSE"
        ORGANIZATION INDEXED
        ACCESS MODE RANDOM
        RECORD KEY IS CRS-ID.
        SELECT STUD-FILE ASSIGN TO "STUDENT".
        SELECT ENROL-FILE ASSIGN TO "ENROLL".
```

```cobol
        DATA DIVISION.
        FILE SECTION.
        FD  CRS-FILE.
        01  CRS-REC.
            02 CRS-ID    PIC X(6).
            02 FILLER    PIC X(65).
02 UNITS      PIC 9.
        FD   STUD-FILE.
        01   STUD-REC.
            02   ROLL-NO       PIC 9(7).
 02  FILLER          PIC X(55).
            02  UNITS-DONE    PIC 99.
            02  CPI           PIC 9V99.
        FD  ENROL-FILE.
        01  ENROL-REC.
            02 EN-ROLL-NO  PIC 9(7).
            02 EN-CRS-ID   PIC X(6).
            02 GRADE       PIC X.
        WORKING-STORAGE SECTION.
        77  WS-ROLL-NO        PIC 9(7).
        77  WS-CRS-ID         PIC X(6).
        77  WS-UNITS-REGD  PIC 99 VALUE 0.
        77  E-O-F          PIC 9 VALUE 0.
        88 STUD-FILE-END    VALUE 1.
        88 CRS-FILE-END     VALUE 2.
        88 ENROL-FILE-END   VALUE 3.
        77  FLAG             PIC 9.
        88 OVER VALUE 1.
        PROCEDURE DIVISION.
        MAIN-PARA.
            OPEN INPUT STUD-FILE.
```

```
        OPEN INPUT CRS-FILE.

        OPEN OUTPUT ENROL-FILE.

        PERFORM  PROCESSING
            UNTIL STUD-FILE-END.

        CLOSE STUD-FILE.

        CLOSE CRS-FILE.

        CLOSE ENROL-FILE.

        STOP RUN.

    PROCESSING.

        PERFORM GET-STUDENT.

        IF NOT STUD-FILE-END
            MOVE 0 TO FLAG
            PERFORM REGISTER-COURSES
                UNTIL OVER.
            DISPLAY WS-UNITS-REGD.

GET-STUDENT.

        READ STUD-FILE RECORD AT END
            MOVE 1 TO E-O-F.

    REGISTER-COURSES.

        DISPLAY "CRSNO:".

        ACCEPT WS-CRS-ID.

        IF WS-CRS-ID = '0'
            MOVE 1 TO FLAG

        ELSE
            MOVE WS-CRS-ID TO CRS-ID
            READ CRS-FILE
            COMPUTE WS-UNITS-REGD = WS-UNITS-REGD + UNITS
            MOVE ROLL-NO TO EN-ROLL-NO
            MOVE WS-CRS-ID TO EN-CRS-ID
            MOVE 'X' TO GRADE
            WRITE ENROL-REC.
```

"COURSE" as viewed by report.cbl & enrol.cbl

```
report.cbl crs_file {
        crs_rec_crs_id     s 6   k
        crs_rec_crs_name   s 40  n
        crs_rec_instr      s 25  n
        crs_rec_filler_1   z 1 n
}


enrol.cbl crs_file {
        crs_rec_crs_id     s 6   k
        crs_rec_filler_1   z 65  n
        crs_rec_units      i 1   n
}
```

"STUDENT" as viewed by report.cbl & enrol.cbl

```
report.cbl stud_file {
        stud_rec_roll_no    i 7   n
        stud_rec_stud_name  s 25  n
        stud_rec_addr       s 30  n
        stud_rec_filler_1   z 5   n
}


enrol.cbl stud_file begin {
        stud_rec_roll_no    i 7   n
        stud_rec_filler_1   z 55  n
        stud_rec_units_done i 2   n
        stud_rec_cpi        f 3.2 n
}
```

"ENROLL" as viewed by enrol.cbl

```
enrol.cbl enrol_file {
```

```
                enrol_rec_en_roll_no i 7   n
                enrol_rec_en_crs_id  s 6   n
                enrol_rec_grade        s 1   n
      }
```

Tables in the recovered Data model are

```
        table course
        (
        crs_rec_crs_id      char(7) not null,
        crs_rec_crs_name    char(41),
        crs_rec_instr       char(26),
        crs_rec_units       integer
        );


        table student
        (
        stud_rec_roll_no    integer,
        stud_rec_stud_name  char(26),
        stud_rec_addr       char(31),
        stud_rec_units_done integer,
        stud_rec_cpi        float
        );


        table enroll
        (
        enrol_rec_en_roll_no integer,
        enrol_rec_en_crs_id  char(7),
        enrol_rec_grade      char(2)
        );
```

Appendix D

Example programs: Set 2

This appendix provides a set of four COBOL programs, and the recovered data model from these programs.

Program "add.cbl"

```
        IDENTIFICATION DIVISION.
        ENVIRONMENT DIVISION.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
                SELECT MASTER-FILE ASSIGN "MASTER.DAT"
                ORGANIZATION INDEXED
                ACCESS DYNAMIC
                RECORD KEY UNAME.
                SELECT I-FILE ASSIGN TO "BUDGET.DAT"
                ORGANIZATION LINE SEQUENTIAL.
        DATA DIVISION.
        FILE SECTION.
        FD MASTER-FILE.
        01 MASTER-REC.
                03 UNAME        PIC X(8).
                03 GRPNO        PIC 9(4).
```

```
            03 UGRP        PIC X(8).
            03 UNUM        PIC X(7).
            03 FILLER      PIC X(82).
            03 UBGT        PIC 9(7).
FD I-FILE.
01 I-REC.
02 V0 PIC X(8).
WORKING-STORAGE SECTION.
77 V2 PIC 9(7).
77 I PIC 999.
77 DD PIC 99.
77 TCHG PIC 9(8).
PROCEDURE DIVISION.
MAIN.
        OPEN I-O MASTER-FILE.
        OPEN INPUT I-FILE.
        PERFORM LOOP.
LAST-PARA.
        CLOSE MASTER-FILE I-FILE.
        STOP RUN.
LOOP.
        READ I-FILE AT END GO TO LAST-PARA.
        MOVE V0 TO UNAME.
        READ MASTER-FILE INVALID DISPLAY UNAME " NOT FOUND"
        GO TO LOOP.
        ADD 2000 TO UBGT.
        ADD UCHG UDUES UPGCHG UPTCHG ULSCHG GIVING TCHG.
        IF TCHG GREATER THAN UBGT
        DISPLAY "NAME:"UNAME ": BUDGET IS INSUFFICIENT ?"
        ELSE DISPLAY "ADDING BUDGET FOR: "UNAME.
        REWRITE MASTER-REC INVALID DISPLAY "NOT ENTERED "UNAME .
```

```
            GO TO LOOP.


Program "addbusget.cbl"



        IDENTIFICATION DIVISION.
        ENVIRONMENT DIVISION.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
                SELECT MASTER-FILE ASSIGN "MASTER.DAT"
                ORGANIZATION INDEXED
                ACCESS DYNAMIC
                RECORD KEY UNAME.
                SELECT I-FILE ASSIGN TO "BUDGET.DAT"
                ORGANIZATION LINE SEQUENTIAL.
                SELECT E-FILE ASSIGN TO "UBDGT"
                ORGANIZATION LINE SEQUENTIAL.
        DATA DIVISION.
        FILE SECTION.
        FD MASTER-FILE.
        01 MASTER-REC.
                03 UNAME        PIC X(8).
                03 GRPNO        PIC 9(4).
                03 UGRP         PIC X(8).
                03 UNUM         PIC X(7).
                03 WEEK-CHG         OCCURS 5 TIMES PIC 9(7).
                03 FILLER       PIC X(47).
                03 UBGT         PIC 9(7).
        FD I-FILE.
        01 I-REC.
        02 V0 PIC X(8).
        02 FILLER PIC X.
```

```
        WRITE O-REC
        ADD UCHG UDUES UPGCHG UPTCHG ULSCHG GIVING TCHG.
        SUBTRACT TCHG FROM UBGT GIVING TEMP.
        IF TEMP < 0
        DISPLAY "        NAME:"UNAME "-VE BALANCE : "TEMP
        ELSE
        DISPLAY "        NAME:"UNAME "+VE BALANCE : "TEMP.
        REWRITE MASTER-REC INVALID DISPLAY "NOT ENTERED "UNAME .
        GO TO LOOP.
```

Program "caddbudget.cbl"

```
    IDENTIFICATION DIVISION.
    ENVIRONMENT DIVISION.
    INPUT-OUTPUT SECTION.
    FILE-CONTROL.
        SELECT MASTER-FILE ASSIGN "MASTER.DAT"
        ORGANIZATION INDEXED
        ACCESS DYNAMIC
        RECORD KEY UNAME.
        SELECT I-FILE ASSIGN TO "BUDGET.DAT"
        ORGANIZATION LINE SEQUENTIAL.
        SELECT E-FILE ASSIGN TO "UBDGT"
        ORGANIZATION LINE SEQUENTIAL.
    DATA DIVISION.
    FILE SECTION.
    FD MASTER-FILE.
    01 MASTER-REC.
        03 UNAME        PIC X(8).
        03 GRPNO        PIC 9(4).
        03 UGRP         PIC X(8).
        03 FILLER       PIC X(54).
```

```cobol
            03 UPGCHG          PIC 9(7).
            03 UPTCHG          PIC 9(7).
            03 ULSCHG        PIC 9(7).
            03 UCHG          PIC 9(7).
            03 UDUES           PIC 9(7).
            03 UBGT          PIC 9(7).
FD I-FILE.
01 I-REC.
02 VO PIC X(8).
02 FILLER PIC X.
02 V1 PIC 9(2).
FD E-FILE.
01 O-REC.
02 O-UNAME PIC X(8).
02 FILLER PIC XX.
02 O-UBGT PIC 9(7).
WORKING-STORAGE SECTION.
77 V2 PIC 9(7).
77 I PIC 999.
77 DD PIC 99.
77 TCHG PIC 9(8).
77 TEMP PIC 9(7).
PROCEDURE DIVISION.
MAIN.
        OPEN I-O MASTER-FILE.
        OPEN INPUT I-FILE.
        OPEN EXTEND E-FILE.
        PERFORM LOOP.
LAST-PARA.
        CLOSE MASTER-FILE I-FILE E-FILE.
        STOP RUN.
```

```
LOOP.
        READ I-FILE AT END GO TO LAST-PARA.
        MOVE V0 TO UNAME.
        READ MASTER-FILE INVALID
        DISPLAY "               "UNAME " NOT FOUND"
        GO TO LOOP.
        MULTIPLY V1 BY 1000 GIVING V2.
        ADD V2 TO UBGT.
        MOVE UNAME TO O-UNAME
        MOVE V2 TO O-UBGT
        WRITE O-REC
        ADD UCHG UDUES UPGCHG UPTCHG ULSCHG GIVING TCHG.
        SUBTRACT TCHG FROM UBGT GIVING TEMP.
        IF TEMP < 0
        DISPLAY "        NAME:"UNAME "-VE BALANCE : "TEMP
        ELSE
        DISPLAY "        NAME:"UNAME "+VE BALANCE : "TEMP.
        REWRITE MASTER-REC INVALID
                   DISPLAY "NOT ENTERED "UNAME .
        GO TO LOOP.
```

Program "subbgt.cbl"

```
        IDENTIFICATION DIVISION.
        ENVIRONMENT DIVISION.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
                SELECT MASTER-FILE ASSIGN "MASTER.DAT"
                ORGANIZATION INDEXED
                ACCESS DYNAMIC
                RECORD KEY UNAME.
                SELECT I-FILE ASSIGN TO "BUDGET.DAT"
```

66

```
                    ORGANIZATION IS LINE SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD MASTER-FILE.
01 MASTER-REC.
            03 UNAME        PIC X(8).
            03 FILLER       PIC X(101).
            03 UBGT         PIC 9(7).
FD I-FILE.
01 I-REC.
02 VO PIC X(8).
02 FILLER PIC X.
02 V1 PIC 9(2).
02 FILLER PIC X.
WORKING-STORAGE SECTION.
77 V2 PIC 9(7).
77 I PIC 999.
77 DD PIC 99.
PROCEDURE DIVISION.
MAIN.
            OPEN I-O MASTER-FILE.
            OPEN INPUT I-FILE.
            PERFORM LOOP.
LAST-PARA.
            CLOSE MASTER-FILE I-FILE.
            STOP RUN.
LOOP.
            READ I-FILE AT END GO TO LAST-PARA.
            MOVE VO TO UNAME.
            READ MASTER-FILE INVALID
                    DISPLAY UNAME " NOT FOUND"
```

```
            GO TO LOOP.
            MULTIPLY V1 BY 1000 GIVING V2.
            SUBTRACT V2 FROM UBGT.
            REWRITE MASTER-REC INVALID
                        DISPLAY "NOT ENTERED "UNAME .
            GO TO LOOP.
```

View of "master.dat" as seen by different programs in the set:-

```
add.cbl master_file {
        master_rec_uname   s 8   k
        master_rec_grpno   i 4   n
        master_rec_ugrp    s 8   n
        master_rec_unum    s 7   n
        master_rec_filler_1   z 82 n
        master_rec_ubgt    i 7   n
}


addbudget.cbl master_file {
        master_rec_uname   s 8   k
        master_rec_grpno   i 4   n
        master_rec_ugrp    s 8   n
        master_rec_unum    s 7   n
        master_rec_week_chg1   i 7 n
        master_rec_week_chg2   i 7 n
        master_rec_week_chg3   i 7 n
        master_rec_week_chg4   i 7 n
        master_rec_week_chg5   i 7   n
        master_rec_filler_1   z 47 n
        master_rec_ubgt    i 7   n
}
```

```
caddbudget.cbl master_file {
        master_rec_uname  s 8   k
        master_rec_grpno  i 4   n
        master_rec_ugrp   s 8   n
        master_rec_filler_1 z 54 n
        master_rec_upgchg i 7   n
        master_rec_uptchg i 7   n
        master_rec_ulschg i 7   n
        master_rec_uchg   i 7   n
        master_rec_udues  i 7   n
        master_rec_ubgt   i 7   n
}

subbgt.cbl master_file {
        master_rec_uname  s 8   k
        master_rec_filler_1 z 101 n
        master_rec_ubgt   i 7   n
}
```

View of "budget.dat" as seen by different programs in the set:-

```
add.cbl i_file {
        i_rec_v0 s 8   n
}

addbudget.cbl i_file {
        i_rec_v0  s 8   n
        i_rec_filler_1 z 1 n
        i_rec_v1  i 2   n
}
```

```
caddbudget.cbl i_file {
            i_rec_v0   s 8   n
            i_rec_filler_1   z 1 n
            i_rec_v1   i 2   n
}


subbgt.cbl i_file {
            i_rec_v0   s 8   n
            i_rec_filler_1 z 1 n
            i_rec_v1   i 2   n
            i_rec_filler_2 z 1 n
}
```

View of "udbgt" as seen by different programs in the set:-

```
addbudget.cbl e_file {
            o_rec_o_uname   s 8   n
            o_rec_filler_1 z 2 n
            o_rec_o_ubgt   i 7   n
}


caddbudget.cbl e_file {
            o_rec_o_uname   s 8   n
            o_rec_filler_1 z 2 n
            o_rec_o_ubgt i 7   n
}
```

Tables in the recovered Data model:-

```
table master.dat
(
master_rec_uname         char(9) not null,
```

70

```
master_rec_grpno        integer,
master_rec_ugrp         char(9),
master_rec_unum         char(8),
master_rec_week_chg1    integer,
master_rec_week_chg2    integer,
master_rec_week_chg3    integer,
master_rec_week_chg4    integer,
master_rec_week_chg5    integer,
master_rec_filler_1     char(13),
master_rec_upgchg       integer,
master_rec_uptchg       integer,
master_rec_ulschg       integer,
master_rec_uchg         integer,
master_rec_udues        integer,
master_rec_ubgt         integer
)

table budget
(
o_rec_o_uname           char(9),
o_rec_filler_1          char(3),
o_rec_o_ubgt            integer
)
```

Bibliography

[AMR94] P. Aiken, A. Muntz, and R. Richards. Dod legacy systems reverse engineering data requirements. *Communications of the ACM*, May 1994.

[ASU87] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, and tools.* Reading, Mass.:Addison-Wesley, 1987.

[CCMJ92] J.H. Cross, E.J. Chikosfy, and C.H. May Jr. Reverse engineering. *Advances in Computers, Volume 35. Academic press, Inc.*, 1992.

[Dat90] J. Date. *An Introduction to Database Systems.* Narosa publishing house, 1990.

[EKN94] A. Engberts, W.V. Kozaczynski, and J.Q. Ning. Automated support for legacy code understanding. *Communications of the ACM*, May 1994.

[EM93] H.M. Edwards and M. Malcom. Recast: Reverse engineering from cobol to ssadm specification. *Proceedings in Software Engineering*, 1993.

[EN94] Elmasri and Navathe. *Fundamentals of Database Systems.* Addison-Wesley publishing company, 1994.

[PK78] A.S. Phillippakis and L.J. Kajmier. *Information Systems through COBOL.* McGraw-Hill, 1978.

[RD89] M.K. Roy and Datidar D.G. *COBOL programming.* Tata McGraw-Hill, 1989.

[SK96] A. Satish Kumar. *Re-engineering: COBOL to C-SQL.* M.Tech Thesis, Department of Computer Science, IIT, Kanpur, February 1996.

[Sri93]   P.L. Srinivas. *On Re-engineering COBOL programs*. M.Tech Thesis, Department of Computer Science, IIT, Kanpur, March 1993.

[SS95]   S. Subramanya Sastry. *Re-engineering COBOL programs*. B.Tech Thesis, Department of Computer Science, IIT, Kanpur, April 1995.

[Ull92]   D.J. Ullman. *principles of Database Systems*. Galgotia publication pvt. ltd, 1992.

A121738